NASA Technical Memorandum 88236

Development of an Interface for an Ultrareliable Fault-Tolerant Control System and an Electronic Servo-Control Unit

Charles Shaver and Michael Williamson

September 1986

```
(NASA-TM-88236)  DEVELOPMENT OF AN INTERFACE          N88-13367
FOR AN ULTRARELIABLE FAULT-TOLERANT CONTROL
SYSTEM AND AN ELECTRONIC SERVO-CONTROL UNIT
(NASA)  37 p  Avail: NTIS HC  A03/MF  A01             Unclas
                                   CSCL 14B G3/09    0113864
```

NASA

National Aeronautics and
Space Administration

NASA Technical Memorandum 88236

Development of an Interface for an Ultrareliable Fault-Tolerant Control System and an Electronic Servo-Control Unit

Charles Shaver,
Michael Williamson, Ames Research Center, Moffett Field, California

September 1986

NASA
National Aeronautics and
Space Administration

Ames Research Center
Moffett Field, California 94035

SUMMARY

The NASA Ames Research Center sponsors a research program for the investigation of Intelligent Flight Control Actuation Systems. The use of artificial intelligence techniques in conjunction with algorithmic techniques for autonomous, decentralized fault management of flight-control actuation systems will be explored under this program. This paper documents the design, development, and operation of the interface and emulator equipment used for laboratory investigations of this research program. The interface, architecturally based on the Intel 8751 microcontroller, is an interrupt-driven system designed to receive a digital message from an ultrareliable fault-tolerant control system (UFTCS). The interface links the UFTCS to an electronic servo-control unit, which controls a set of hydraulic actuators. It was necessary to build a UFTCS emulator (also based on the Intel 8751) to provide signal sources for testing the equipment.

This paper discusses the conversion of the 8-byte message (characteristic command for the four control axes of a helicopter) to the appropriate byte length.

INTRODUCTION

The NASA Ames Research Center sponsors a research program for the investigation of intelligent flight-control-actuation systems. This paper documents the design, development, and operation of the interface and emulator equipment used for laboratory investigations of the research program.

The interface was designed to receive a digital message from an ultrareliable fault-tolerant control system (UFTCS), a quadruplex asynchronous microprocessor system, and output data to an electronic servo-control unit (ESCU), an electronic controller for a set of hydraulic actuators. Since the UFTCS was not available for testing the interface, a UFTCS emulator was designed and built for testing purposes. The interface can also transmit a feedback message from the ESCU to the UFTCS upon request from the UFTCS. The digital message the interface receives and transmits contains a minimum of 12 bytes and a maximum of 76 bytes. The main feature of the message is the 8 bytes that describe the command for the four control axes of a helicopter.

The architecture of the interface is based on the Intel 8751 microcontroller and is an interrupt-driven system. The 8751 controls the flow and distribution of data for receiving messages from the UFTCS and transmitting messages to the UFTCS. This paper describes in detail the flow of data from the UFTCS through the interface to the ESCU (receiving messages) and the data flow from the ESCU through the interface to the UFTCS (feedback message). The hardware for each case is discussed. The software is also described. The various operating modes and calibration routines are described, as in the interrupt structure developed for the Intel 8751. Finally, procedures for operating the interface and the emulator together are listed.

The UFTCS emulator was built to give the interface a signal source because the UFTCS was not available. The emulator is also a design based on the 8751. The primary use of the emulator is to provide a digital message for the interface to receive and process. The variable update rates of the emulator make it possible to study the effect of update rate on quantization. Attempts to smooth the quantization can be made in the interface during the time between updates because no interrupts are occurring. The 8-bit architecture of the 8751 makes the smoothing algorithms cumbersome because the data for the control axes from the UFTCS are transmitted at 16 bits/sec. Because all mathematics and shifting routines are not supported by the instruction set of the 8-bit 8751, separate routines must be written to accomplish any calculations that must be performed to smooth the output of the interface. The development of 16-bit microcontrollers will greatly simplify the algorithms necessary to reduce quantization effects.

UFTS/ESCU INTERFACE DESCRIPTION

The interface developed links the UFTCS to an ESCU. The UFTCS is a quad-redundant, flight-control-law processor that is based on an asynchronous architecture (ref. 1) and designed for testing on the UH-1H helicopter. The ESCU controls the series hydraulic actuators for stability augmentation. The interface is also a quad-redundant system. A fully operational interface can receive four channels of data from the UFTCS and transmit four sets of four command signals to the ESCU. When feedback is requested, the interface can transmit the latest position of the actuatcrs to the UFTCS.

The interface receives digital messages from the UFTCS system. Messages are sent from the UFTCS through differential signal lines or fiber optic cables. The analog signal is sent to the ESCU using coaxial cables; each message is made up of 10 to 76 bytes of information. Figure 1 shows the message format for a message from the UFTCS.

The first byte of data is the message byte count, which tells the interface how many bytes of information will be transmitted. The message byte count is followed by the subblock 1 byte count. Each message is divided into two subblocks. At this time, subblock 2 is not used and has a count of zero.

The subblock 1 byte count is followed by the subblock 1 data which consist of the aircraft commands and a block of data called "extra" data. The first eight bytes of subblock 1 contain the aircraft commands that control the four flight axes of the aircraft. Each command is made up of two bytes. The low byte is always received before the high byte.

MESSAGE BYTE COUNT	SUBBLOCK 1 BYTE COUNT	ROLL COMMAND LOW BYTE	ROLL COMMAND HIGH BYTE	PITCH COMMAND LOW BYTE	PITCH COMMAND HIGH BYTE

YAW COMMAND LOW BYTE	YAW COMMAND HIGH BYTE	COLLECTIVE COMMAND LOW BYTE	COLLECTIVE COMMAND HIGH BYTE	EXTRA DATA	SUBBLOCK 2 BYTE COUNT

CHECK SUM

Figure 1.- Message from UFTCS format.

The extra data serve no purpose at this time. The subblock 2 byte count which follows the extra data is given the number zero because no information has been placed in subblock 2. The final byte sent is a checksum. The checksum can be compared to that checksum formed by the interface as data were received, which allows for message verification. However, no verification of the checksum check has been included in this system, and no software has been designed to retransmit a bad message.

Another function of the interface is to send feedback data to the UFTCS. The feedback variable is the latest position of the actuators in the aircraft. The feedback data are sent only after a request for data from the UFTCS has been received by the interface. When a request for data has been received, the interface reads the voltages on the feedback lines and then sends the feedback data back to the UFTCS in digital form.

INTERFACE HARDWARE: DATA FROM THE UFTCS

The interface is made up of five boards. One board is called the signal-distribution board. This board receives the digital messages from UFTCS and distributes the messages to the remaining four boards, called the signal-processing boards. The signal-distribution board can receive signals in optical or differential signal form. To date, only differential signals are sent to the interface.

The hardware required on the signal-distribution board to receive messages from the UFTCS and transmit feedback requires the following equipment.

1. Four receivers to receive messages from the UFTCS
2. One receiver to receive a request for data from the UFTCS
3. One transmitter to transmit a sync pulse to the UFTCS at the start of a feedback transmission

4. One transmitter to transmit feedback messages

The above circuitry has an optical mode and a differential signal mode to accommodate optical or differential signals.

The four signal-processing boards all contain the circuitry to receive data from the UFTCS. One of these four boards contains the extra circuitry needed to handle the feedback function of the interface. Feedback to the UFTCS represents the most recent position of the actuators. Figure 2 shows the flow of data from the UFTCS through the interface to the ESCU.

The key components used on the signal-processing boards when receiving a message from the UFTCS are listed below.

1. Intel 8751 microcontroller
2. Harris 6402A Universal Asynchronous Receiver Transmitter (UART)
3. Harris Programmable Peripheral Interface (PPI)
4. Digital-to-Analog Converters (DACs)

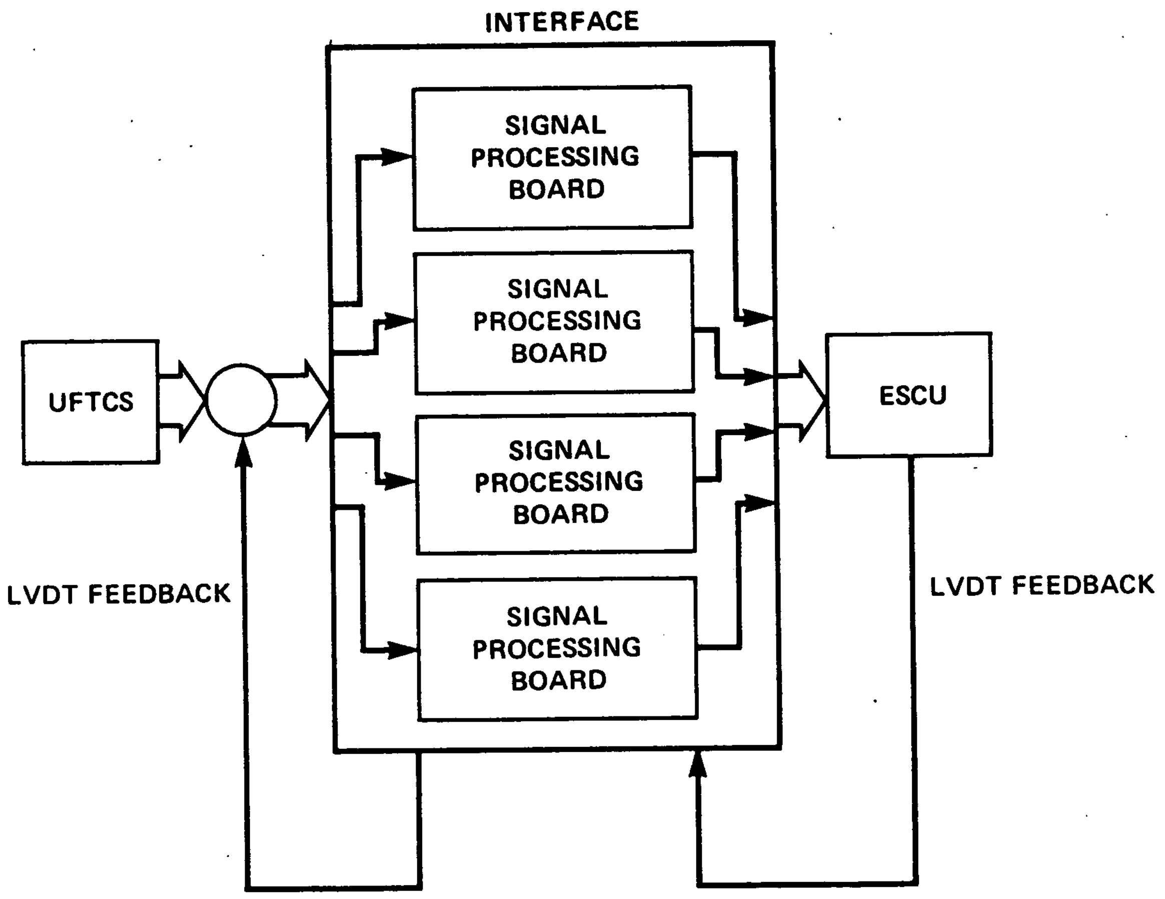


Figure 2.- Data flow from UFTCS to ESCU.

Intel 8751

The Intel 8751 is an erasable programmable read only memory (EPROM) version of the 8051 family of 8-bit microcontrollers. The crystal used to run the 8751 operates at a frequency of 10 MHz. The 8751 is an input/output (I/O) controller. The main features of the 8751 listed below are explained in the paragraphs that follow.

1. Four 8-bit I/O ports
2. Two external interrupts
3. A serial I/O port
4. Two internal timers

Input-Output Ports- The four I/O ports are numbered 0-3. In addition to being an I/O port, port 3, which contains the serial port and the external interrupts, can be used as a special-functions register.

External Interrupts- The two external interrupts are used for signaling the 8751 that a message from the UFTCS has been received or that a request for data has been sensed. The interface is capable of receiving and transmitting messages to and from the UFTCS using a UART or the serial port of the 8751. If a UART is used, one of the external interrupts is used to signal the 8751 that the message is in the UART and is ready for the 8751 to process. The second external interrupt is only used on the signal-processing board that contains the feedback circuitry. This interrupt is triggered when the UFTCS requests feedback data. Reception of an external interrupt, or any interrupt, will cause the 8751 to jump to a software routine written to service the interrupt that preempted the message being processed by the 8751.

Serial Port- The serial port can also be used to receive and transmit information to and from the UFTCS. The baud rate of the serial port is set by the software, and for this application is 156K baud (156,000 bits/sec). The serial port interrupt is internal to the 8751. This interrupt is set upon reception of a stop bit when receiving data or transmission of a stop bit when transmitting data.

The serial port interrupt is triggered whether the 8751 is receiving or transmitting data. Once the 8751 has jumped to the serial port interrupt routine, a further check must be made to determine if the interrupt was caused by receiving data or transmitting data. After determining the type of interrupt the interface software then branches to the proper routine to service the interrupt.

Timers- One of the internal timers is configured as a 16-bit timer and is used in one of the testing routines of the 8751. The other timer is configured as an 8-bit timer and serves two purposes.

The first purpose of the 8-bit timer is to monitor while the 8751 is in run mode by guarding against an undetected end of message from the UFTCS. The second use for the timer is to test the DACs by cycling the DACs through their output range in 1 min.

Universal Asynchronous Receiver/Transmitter

The UART is used on the signal-processing board to transfer data from the signal-distribution board to the 8751. The UART receives data in serial form and sends it to the 8751 in parallel form. When the UART has received data from the signal-distribution board, it sends an interrupt pulse to the 8751 on one of the external interrupts.

If the UART is transmitting a feedback message to the UFTCS, the 8751 will send the data to the UART in parallel. The UART adds the start, stop, parity bits and then transmits the data, in serial, to the signal-distribution board. The signal-distribution board transmits the message, in serial, to the UFTCS.

The UART uses a crystal that operates at 15 MHz which sets the baud rate of the UART at 156K baud. The UART and the serial port of the 8751 have the same baud rate, which allows operation of the board using a UART or the serial port of the 8751. This option is selected by setting one of the dip switches located next to the 8751.

Programmable Peripheral Interface

The PPI receives data from the 8751, then writes the data to the DACs. The PPI has three 8-bit ports, called A, B, and C, that can be programmed to operate in three different modes, which are numbered 0, 1, and 2. The PPI used to receive messages from the UFTCS has been programmed to operate in mode 0 such that ports A and B are output ports, while 4 bits of port C are output and 4 bits are input.

The low byte of each command word is sent using port A, while the high byte of each command is sent using port B. A latch command is sent using port C. The 16-byte command words received from the UFTCS are written to the DACs.

Digital to Analog Converter

The DACs receive the 16-bit words and convert them to analog voltages. The latch command is used to "lock" each word into the DAC, and no new words written to the DACs will be converted unless the latch is removed before the word is written. The DACs have an output range of -10 volts to +10 volts and use offset binary to represent a voltage. The format is shown as follows:

```
0000H = -10 volts
8000H =   0 volts
FFFFH = +10 volts
```

The voltages output by the DACs are the voltages that the ESCU receives.

Figure 3 and the following summary describe the data flow when the interface is receiving a message from the UFTCS.

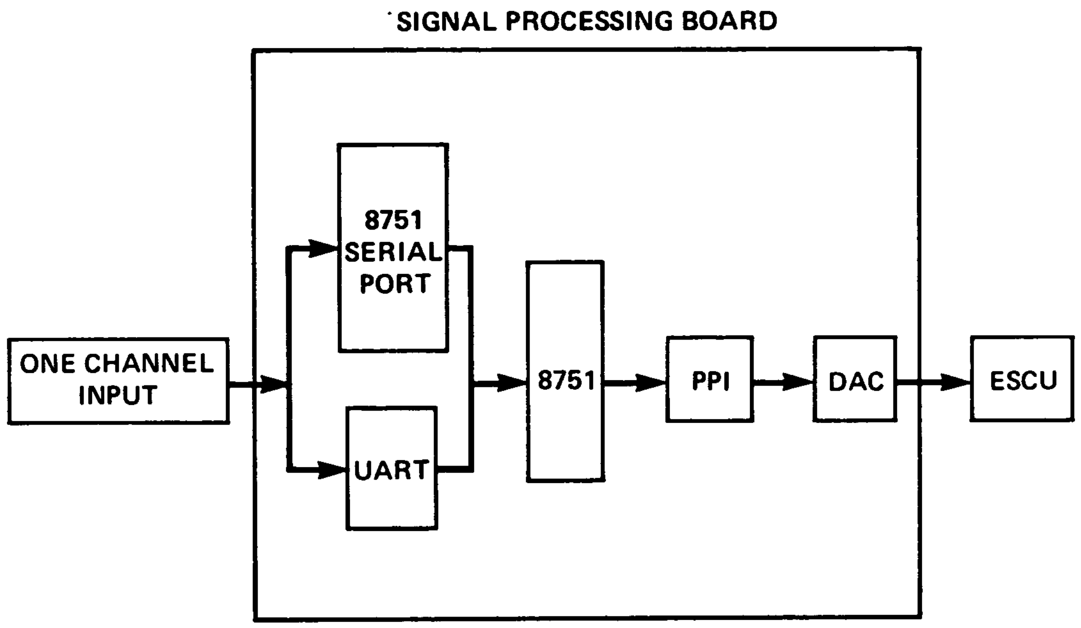


Figure 3.- Data flow through the interface to the ESCU.

The signal reception summary is listed below.

1. Data are received by the UART or serial port of the 8751 from optical fibers or differential signal.
2. An interrupt is sent to the 8751 to service the data.
3. The data are received by the serial port of the 8751 or a UART one byte at a time and sent to the PPI by the 8751.
4. The PPI sends 16-bit words to the DACs.
5. The DACs output analog voltages to the ESCU.

INTERFACE HARDWARE: DATA TO UFTCS

One of the four boards that receive data from the UFTCS contains the extra circuitry to send actuator position feedback to the UFTCS. The key components for the feedback circuitry are shown below and are explained in the order they are listed.

1. Analog Multiplexer
2. Sample/Hold Amplifier
3. Analog to Digital Converter
4. Programmable Peripheral Interface

Analog Multiplexer

The analog multiplexer (analog mux) chooses which channel of feedback will be read. The 8751 signals the analog mux which channel of feedback to look at. The mux reads the channel chosen, and the voltage passes through an operational amplifier (op-amp). The voltage from the op-amp then goes to the sample and hold (S/H) chip.

Sample and Hold

The S/H amplifier passes samples of the voltage from the analog mux to the analog to digital converter (ADC). Upon receiving a hold pulse from the 8751, the S/H amplifier will sample the voltage on its input pin and hold that voltage on its output pin until the next hold pulse is received.

Analog to Digital Converter

The ADC converts the analog feedback signal to a digital representation of the voltage. The hold pulse sent to the S/H chip is also used to trigger the convert pin of the ADC. The pulse used to trigger a conversion is passed through a flip-flop network to delay the arrival of the convert pulse at the ADC. This ensures that the voltage from the S/H chip arrives at the ADC before the conversion pulse.

The ADC outputs a 12-byte representation of the analog voltage. Since each axis command is 16 bytes long, the last 4 bits of the voltage are set to zero. The digital data are then passed to another PPI.

Programmable Peripheral Interface

The PPI used in the feedback circuitry is the same type used in the circuitry to receive data from the UFTCS and is simply programmed to work in a different mode. The feedback PPI configuration is listed as follows:

1. Port A--strobed input
2. Port B--basic input
3. Port C--handshaking/control lines

Ports A and C work together to receive parallel data from the ADC. When the ADC has data to send, it sends a pulse to port C and after port C has received the pulse, the digital data are received in ports A and B. When ports A and B have received data, port A sends a pulse back to the ADC to acknowledge reception of the data. Sending pulses back and forth between components or devices is called handshaking.

Port B is receiving the second half of the message that port A receives. It is not necessary for port B to send a pulse back to the ADC; therefore, port B works as a basic input port.

From the PPI the information passes to the 8751 microcontroller and the digital feedback data are then sent to the UFTCS using the serial port or the UART. Another form of handshaking occurs before the feedback is transmitted to the UFTCS when the 8751 sends out a sync pulse to the UFTCS which signals the UFTCS that a message is about to be transmitted.

Figure 4 and the accompanying summary describe the transmission of feedback data when the UFTCS has issued a request for data as follows:

1. A request for data is sent from the UFTCS.
2. The 8751 receives an interrupt and jumps to a software routine to service the interrupt.
3. The analog mux reads a channel of feedback and sends it to the S/H chip.
4. The S/H chip waits for a hold pulse and then sends voltage to the ADC.
5. The ADC receives a convert pulse and outputs digital data to the PPI.
6. The PPI transfers data to the 8751.
7. A sync pulse is sent by the 8751.
8. Data are sent to the UFTCS using a UART or the serial port of the 8751.

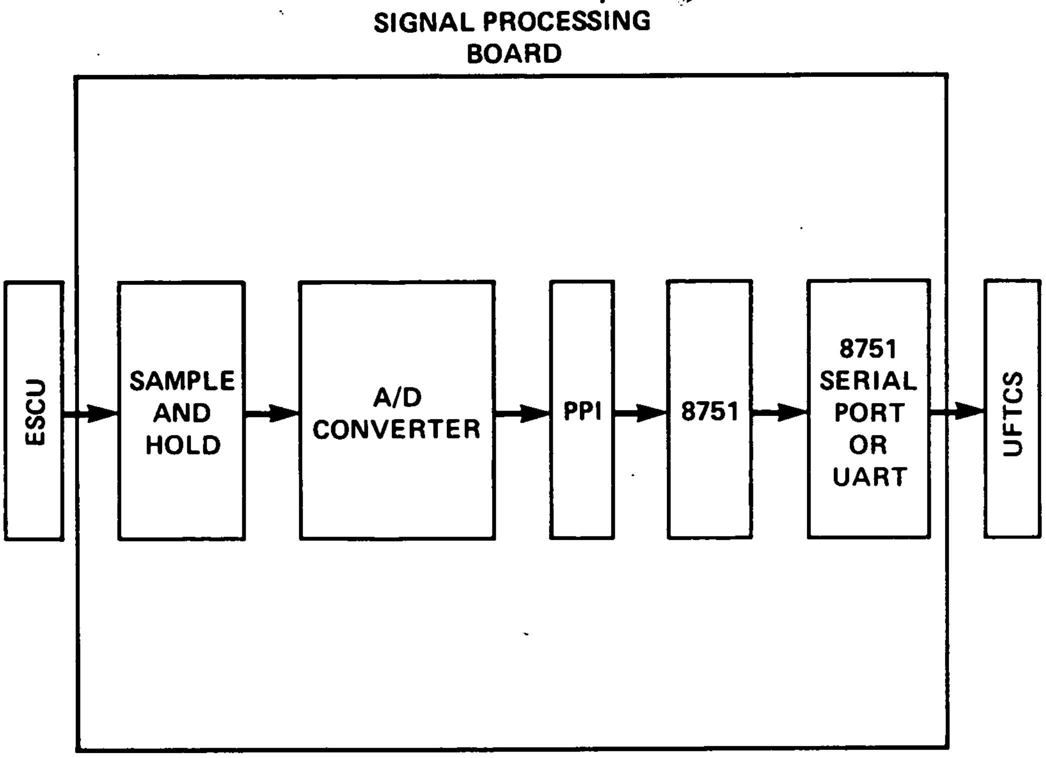


Figure 4.- Data flow through the interface to the UFTCS.

INTERFACE SOFTWARE

The software written for the 8751 was designed to work using the serial port or a UART. The type of board to be used is chosen by the settings of the dip switches located next to the 8751.

The dip switches are also used to select the different operating modes of the board. There are two switches for choosing the type of board and six switches for choosing the operating modes of the 8751. The type of board or operating mode is selected by setting the switch in the off position.

The operating modes listed below are selected with two switches.

1. UART or serial port of the 8751 microcontroller
2. Feedback circuitry or no feedback circuitry

The two switches are combined with the 8751 in such a way as to yield four possible board configurations. These are:

1. Feedback circuitry with UART
2. Feedback circuitry with serial port of the 8751
3. No feedback circuitry with UART
4. No feedback circuitry with serial port of the 8751

Dip switches 6 and 7 are used for selecting the operating mode; it is important to have these switches set correctly for each board. The board-type selection chart which shows the switch settings for each board-type selection is shown in table 1.

TABLE 1.- BOARD-TYPE SELECTION CHART

Switch number	Set	Not set
6	UART	Serial port
7	Feedback circuitry	No feedback circuitry

There are seven operating modes for the interface, one of which is the run mode, which is the mode used under normal operating conditions. The other six modes are calibration and testing modes. The calibration and testing modes are listed below.

1. Interboard loopback test
2. On-board loopback test (for boards with feedback circuitry only)
3. ADC calibration (for boards with feedback circuitry only)
4. DAC gain calibration

5. DAC offset calibration
6. DAC testing routine

The operating modes are chosen on a "first switch set" basis. If a test or calibration is chosen for a board with feedback circuitry, switch 7 must be off for that board. If the test is tried when switch 7 is on, the error light will light. The 8751 looks for the first mode selection switch set and goes into that mode of operation. All remaining switches are ignored. Table 2 shows the priority level for each mode selection and aslo gives the switch associated with each mode. The priorities are listed from the highest to the lowest.

TABLE 2.- OPERATING MODE PRIORITY

Switch set	Operating mode (highest to lowest priority)
8	Interboard loopback test
1	Run mode
2	ADC calibration
3	On-board loopback test
4	DAC gain calibration
5	DAC offset calibration
None set	DAC testing routine

Interboard Loopback Test

The interboard loopback test verifies that the receivers, transmitters, and communication between the boards are all working. The transmitter is connected to one of the four receivers used to receive data from the UFTCS. An analog signal is input as feedback from the ESCU. A request for data signal must be input at the "request for data" pin.

The test is started from the feedback service routine. Since the software was written so that data must be received before it is transmitted, register 5 of register bank 0 is set to 8. This makes the 8751 think it has received some data.

The 8751 will sample the analog signal, convert it to digital, and transmit it over optical fibers or differential signal lines to another board. The board receiving the data will convert the signal back to analog and output the signal on the board that received the digital signal. The output should be a sampled version of the analog feedback signal.

On-board Loopback Test

The on-board loopback test will sample the analog feedback signal input at the ESCU feedback pins at approximately 50-msec intervals. An analog to digital conversion will be done on the signal, and the 16-bit word will be written to all four DACs simultaneously. The output of the DACs will again be a sampled representation of the analog feedback signal.

ADC Calibration

The ADC calibration will cause the 8751 to send a hold pulse to the S/H chip every 33 μsec. The signal on the input to channel one of the analog mux will be sampled by the S/H chip or amplifier. Thus, the input at channel one should be a precision voltage source.

Adjust the voltage source until it reads -9.9999 volts. Adjust the S/H chip potentiometer until it reads -9.9999 volts. Adjust the ADC output until it reads FFEH.

DAC Gain Calibration

The 8751 will write the value 0000H to each DAC. Adjust each DAC gain potentiometer until the DAC output reads -10.0000 volts.

DAC Offset Calibration

The 8751 will write the value 8000H to each DAC. Adjust each DAC offset potentiometer until the DAC output reads 0.0000 volts.

DAC Testing

The DAC test will run the DACs through their output range of -10 volts to +10 volts. Once +10 volts is reached, the DACs will run backward to -10 volts. The run up to +10 volts and back down to -10 volts is one cycle. Timer 1 is used to make a complete cycle last about 2 min. The test verifies that the DACs operate correctly and continues until the the interface has been reset.

Interrupts

The software is designed to be compatible with an interrupt-driven system, which means that the 8751 will do nothing until an interrupt has been sensed. When an interrupt has been sensed, the 8751 will jump to an appropriate software routine to service the interrupt.

The 8751 is designed so that priority levels can be established for the interrupts. An interrupt can be assigned a high or low priority. A low-priority interrupt can be interrupted by a high-priority interrupt, but a high-priority interrupt cannot be interrupted.

Receiving data from the UFTCS is more important than sending feedback to the UFTCS. For this reason, the UART and the receive pin of the serial port are assigned a higher priority. Timer 1 also carries higher priority since it monitors the reception of data from UFTCS.

The type of interrupt scheme used depends on whether a UART or serial port is being used. The interrupt schemes for each type of board are listed in table 3.

TABLE 3.- INTERRUPT SCHEMES

Interrupts for UART	
Interrupt	Priority
*INT1	High
Timer 1	High
*INT0	Low
Timer 0	Low
Interrupts for serial port	
Serial port	High
Timer 1	High
*INT0	Low
Timer 0	Low

Operation with a UART

There are four interrupts active when a UART is used to receive and transmit data from the UFTCS. The interrupt, *INT1, is used to signal the 8751 that a message has been received in the UART and is assigned a high priority so that no incoming information will be lost. Timer 1 monitors incoming messages to guard against an undetected end of message.

The interrupt, *INT0, is used only on boards with feedback circuitry and is triggered when the UFTCS requests feedback data. This interrupt has a low priority so that it can be interrupted if incoming data have just been received.

Timer 0 is used to test the DACs and therefore is not used when the interface is in the run mode. The timer 0 interrupt is assigned a low priority since it is the only interrupt active when in use.

Operation with the Serial Port of the 8751

There are also four interrupts active when the interface is used with the serial port of the 8751. The only difference between the two configurations (serial port or UART) is the interrupt used to signal the 8751 that incoming data have been received. Since the serial port of the 8751 has a "built-in" interrupt, it is not necessary to use *INT1 to signal that an incoming message has been received. This built-in interrupt has high priority so that messages being received have higher priority than feedback messages being transmitted to the UFTCS.

The serial port interrupt routine is designed so that a message being received is serviced before a message being transmitted. The reason for this structure is that the 8751 serial port interrupt is triggered whether the 8751 is receiving or transmitting data. Therefore, a further test must be performed when entering the serial port service routine to ensure that the 8751 enters the correct data-processing routine--either for receiving or transmitting data.

Appendix A contains schematics of the signal-distribution board, the signal-processing board, and the feedback circuitry present on one of the signal-processing boards.

UFTCS EMULATOR

The UFTCS emulator simulates the digital message transmitted by the UFTCS flight-control system. The UFTCS emulator sends messages to the UFTCS/ESCU interface. The emulator is used to test and verify the software written for the interface. It is also used to drive the interface for testing the hydraulic actuator. The message sent by UFTCS has the format shown in figure 5.

The length of a message sent by the UFTCS varies from 10 to 76 bytes. The variation in message length is a function of the number of bytes sent to the interface as extra data. The extra data, if sent, have a length ranging from 1 to 66 bytes.

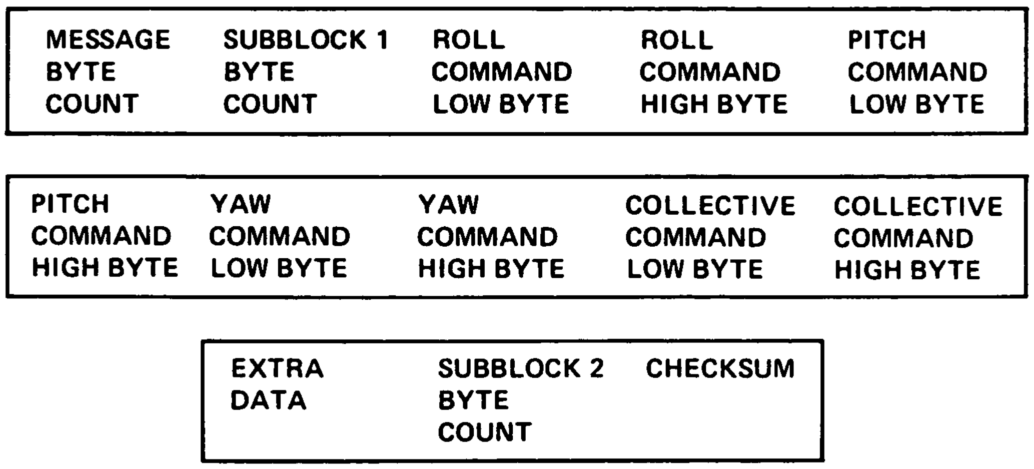


Figure 5.- Emulator message format.

The UFTCS emulator has three important features.

1. Calibration or run mode
2. Variable message update rates
3. Variable message lengths

The desired mode of operation is obtained by setting the dip switches located on the emulator. A switch is set by pushing it toward the open position.

There are two dip switches, each equipped with four switches, on the board. One dip switch contains the calibration and message update rate selection, while the other dip switch contains the message length selection switches.

The calibration and update dip switch contains the switches to set the message update rate and operating mode of the UFTCS emulator. The message update rate selection switches operate on a "first switch set" structure. The software tests each switch and chooses the update rate of whatever switch is set. All remaining switches are then ignored.

Table 4 shows the switch settings for the calibration and update switch and the message update rate switches listed in highest to lowest priority.

The message length dip switch sets the byte length of the UFTCS emulator. These switches also work in a "first switch set" mode. The software polls the switches from highest priority to lowest priority, and uses the message length associated with the first switch set while the other switches are ignored. If no switch is set, the message length for a "no switch set" is used. The message length is varied by sending a different amount of extra data at the end of the message; the extra data are the only part of a UFTCS message that is variable. Table 5 shows the message lengths associated with each switch.

TABLE 4.- CALIBRATION AND UPDATE RATE

Calibration		
Switch number	Switch set	Switch not set
1	ADC calibration	Run mode

Message update rate selection	
Switch number	Message update rate, msec
2	25
3	10
4	9.2
None set	50

TABLE 5.- MESSAGE LENGTH

Switch number	Message length, bytes
1	76
2	36
3	20
None set	10

EMULATOR HARDWARE

The UFTCS emulator consists of 13 chips. The major chips used in the emulator, which are listed below, are discussed in this section.

1. SMP--11 S/H chips
2. Burr-Brown ADC
3. Harris 82C55 PPI
4. Intel 8751 microcontroller

To place the emulator in run mode, move switch 1 of the calibration and update switches away from the open position.

Sample and Hold

The S/H amplifier chip is used to take samples of the analog input signal by tracking this signal until a hold pulse is received from the 8751. Once the hold pulse is received, the voltage present on the input pin of the S/H chip is placed on the output pin of the S/H chip. The voltage on the output pin of the S/H chip is input to the ADC.

Analog to Digital Converter

The ADC receives a convert pulse shortly after the S/H chip places the voltage on the line. The convert pulse must occur after the voltage appears at the input pin of the ADC to allow for the voltage to become stable. After receiving the convert pulse, the ADC outputs a digital representation of the input analog voltage. This particular ADC outputs a 12-bit representation of the voltage and is called a 12-bit converter.

Programmable Peripheral Interface

The ADC does some handshaking with the PPI. When the ADC is finished converting, the status line of the converter is driven low. This high to low transition is passed through an inverter (74LS04) and is input to the clock pin of a D flip-flop (74LS74). The low to high transition that occurs on the flip-flop clock pin places the Q output high and the NOT Q output low. The Q output is tied to an octal latch (Intel 8282) and the NOT Q is tied to the PPI. The low pulse of the NOT Q signals the PPI that data are being sent to the PPI, which sends back a high pulse to respond that the data have been accepted. The high pulse from the PPI clears the flip-flop and puts the Q output low and the NOT Q output high.

The octal latch is used to pass 8 bits to the PPI. The ADC outputs 12 bits, but the PPI reads 16 bits. The high 4 bits of the output of the octal latch are the least significant bits of the ADC output. The low four bits of the octal latch are tied high. These bits are tied high because the 8751 will complement the output before transmitting it to the interface. The output is low and will not add a direct current (DC) offset to the ADC output. In addition to sending a handshake signal to the ADC, the PPI also signals the 8751 that the data are ready to be read. The PPI acts as a middle man between the 8751 and the ADC.

Another function of the PPI is to send the hold pulse, when directed by the 8751, to the S/H chip and the convert pulse to the ADC.

Intel 8751

The 8751, which is the core of the UFTCS emulator, forms the entire message and transmits it through its serial port at a baud rate of 156K. The message byte count and subblock 1 count are the first two bytes transmitted. Next, the 8751 directs the PPI to signal the S/H chip and the ADC to sample the analog input and place a digital representation of the input analog voltage on the line for the 8751 to read from the PPI. This digital representation makes up the next 8 bytes of the message. These 8 bytes represent the roll, pitch, yaw, and collective axes for the aircraft. These 8 bytes are then transmitted out the serial port of the 8751.

Following the 8 bytes of command axes, the extra data are sent, the length of which can be chosen by the user. The amount of extra data sent can be calculated from the message length table and it is always 10 bytes less than the message length.

Table 6 shows the correlation between message length and the amount of extra data sent.

TABLE 6.- EXTRA DATA

Switch number	Extra data, bytes	Message length, bytes
1	66	76
2	26	36
3	10	20
None set	0	10

Once the 8751 has transmitted all the extra data, the subblock 2 count and a checksum are transmitted. Transmitting the checksum signifies the end of a message. The 8751 will delay an amount of time chosen by the user before starting the transmission of the next message the time between messages is listed in table 4. The emulator will continue to transmit messages as long as the emulator is in run mode. Figure 6 summarizes the transmission of a message by the UFTCS emulator.

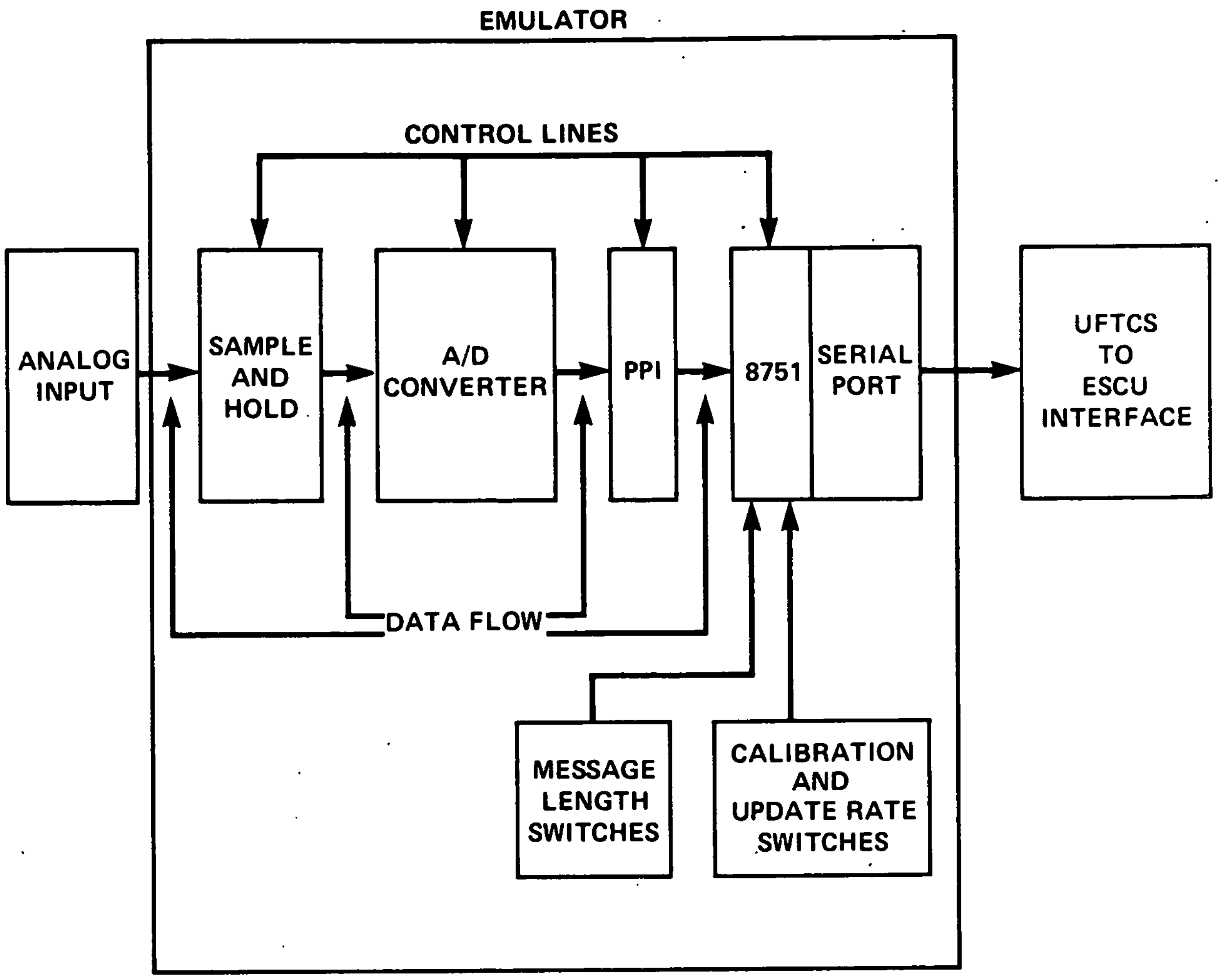


Figure 6.- Emulator message transmission.

OPERATIONAL PROCEDURES

UFTCS Emulator with UFTCS/ESCU Interface

To ensure proper synchronization between the UFTCS emulator and the interface, the following steps must be taken when powering up or resetting the system.

1. If powering up, connect the power cable to the interface and to the emulator. The ends of the power cable are clearly marked.

2. First turn on the 5-volt supply, then the 15-volt supply.

3. Hold the reset switch of the emulator down and flick the reset switch of the interface. Now release the switch of the emulator. This ensures that the

interface is ready to receive a message before the emulator starts sending a message.

If the systems need resetting only, then only the third step needs to be followed.

If the interface has been running with a request-for-data pulse, the function generator supplying the request pulse must be turned off before the systems are reset or an error will occur. (The red light on the interface board that handles feedback will light up immediately after a reset is attempted.)

Analog to Digital Converter Calibration

To calibrate the ADC, switch 1 of the calibration and update switches must be set. The following steps should then be followed:

1. Power up the emulator--5-volt supply first, 15-volt supply second.

2. Attach a precision voltage supply to the emulator analog input and set the voltage to +9.999 volts.

3. Attach a logic analyzer to the output pins of the emulator ADC (12 pins).

4. Flick the reset switch of the emulator and monitor the voltage out of the S/H chip. Set this voltage to +9.999 volts, using the S/SH offset adjustment potentiometer.

5. Adjust the output of the ADC to read 000H using the gain adjust potentiometer.

6. Reverse the polarity of the supply to read -9.999 volts and adjust the output of the ADC to read FFEH using the ADC offset potentiometer.

To return to run mode, switch 1 of the calibration and update switches must be set to the off position and the interface and the emulator reset as described in the previous section.

Appendix B contains a schematic of the UFTCS emulator, as well as a listing of the software written for the 8751 used in the emulator.

CONCLUDING REMARKS

The UFTCS-to-ESCU interface was initially developed as a data interface between the UFTCS (flight-control computer) and the ESCU (controller for the hydraulic actuators).

The emulator was developed to provide a digital message to the interface, was first used to verify its operation, and is now used as a signal source to the interface.

Because the interface has a microprocessor on board, the interface can and has been used to investigate the application of microprocessors to the control of flight-actuation systems. The interface has a limited capability to investigate command smoothing and in-line monitoring models in real time. These algorithms can be executed between message updates since the interface is interrupt-driven. Therefore, these algorithms are only executed when the UFTCS interrupts a message update. The variable update rates of the emulator allow investigations into the effect of update rate upon quantization.

The interface does, however, have limitations as a research tool. Since the 8751 is an 8-bit microcontroller and the commands used are 16 bits, the computations are cumbersome (such as requiring separate subroutines for all mathematical and shifting routines). An interface designed to complement the newer 16-bit microcontrollers would simplify the computations significantly.

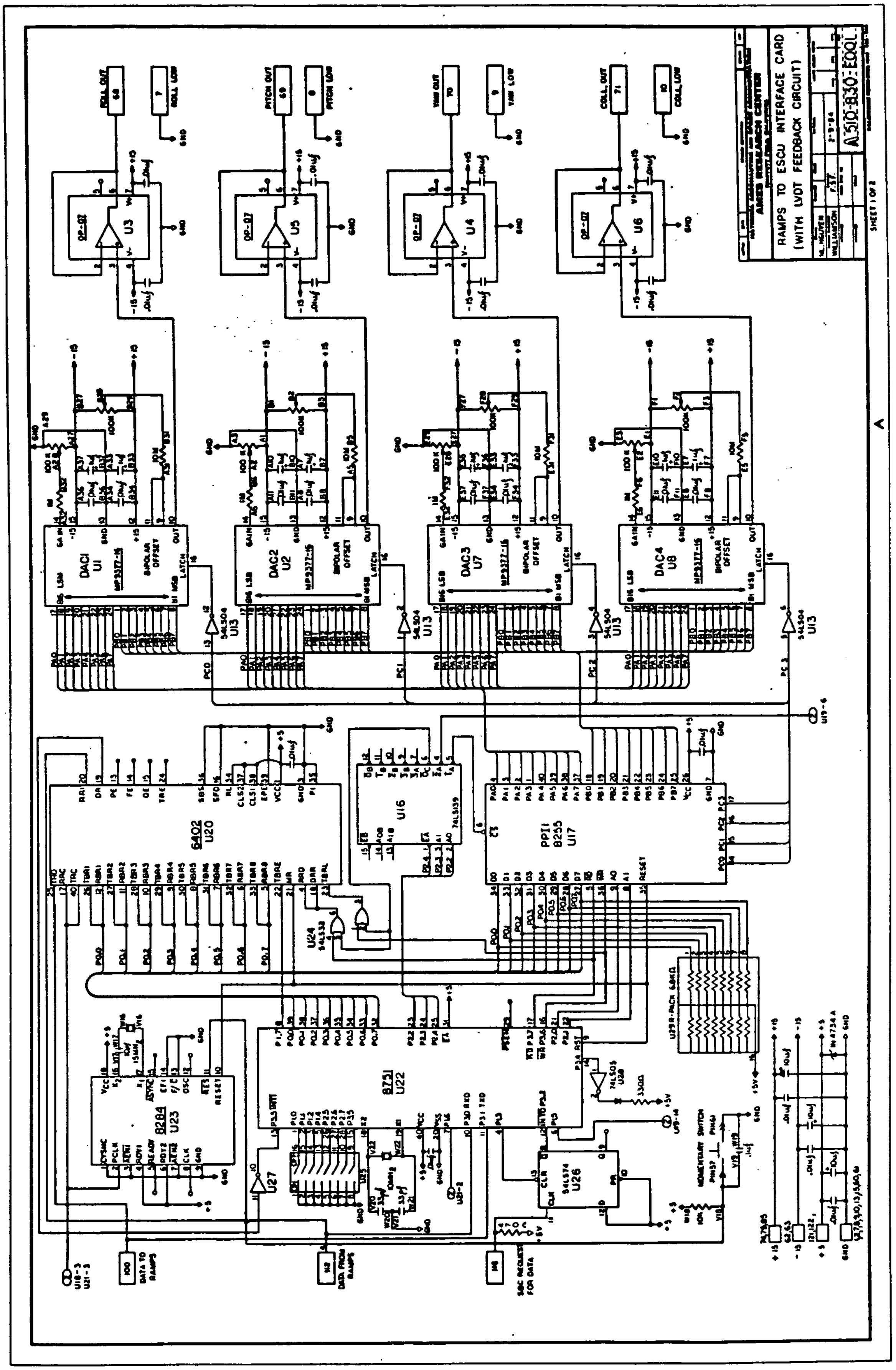
REFERENCES

1. Webster, L. D.; Slykhouse, R. A.; Booth, L. A., Jr.; Carson, T. M.; Davis, G. J.; and Howard, J. C.: Ultrareliable Fault Tolerant Control Systems. AIAA Paper 84-2650, 1984.

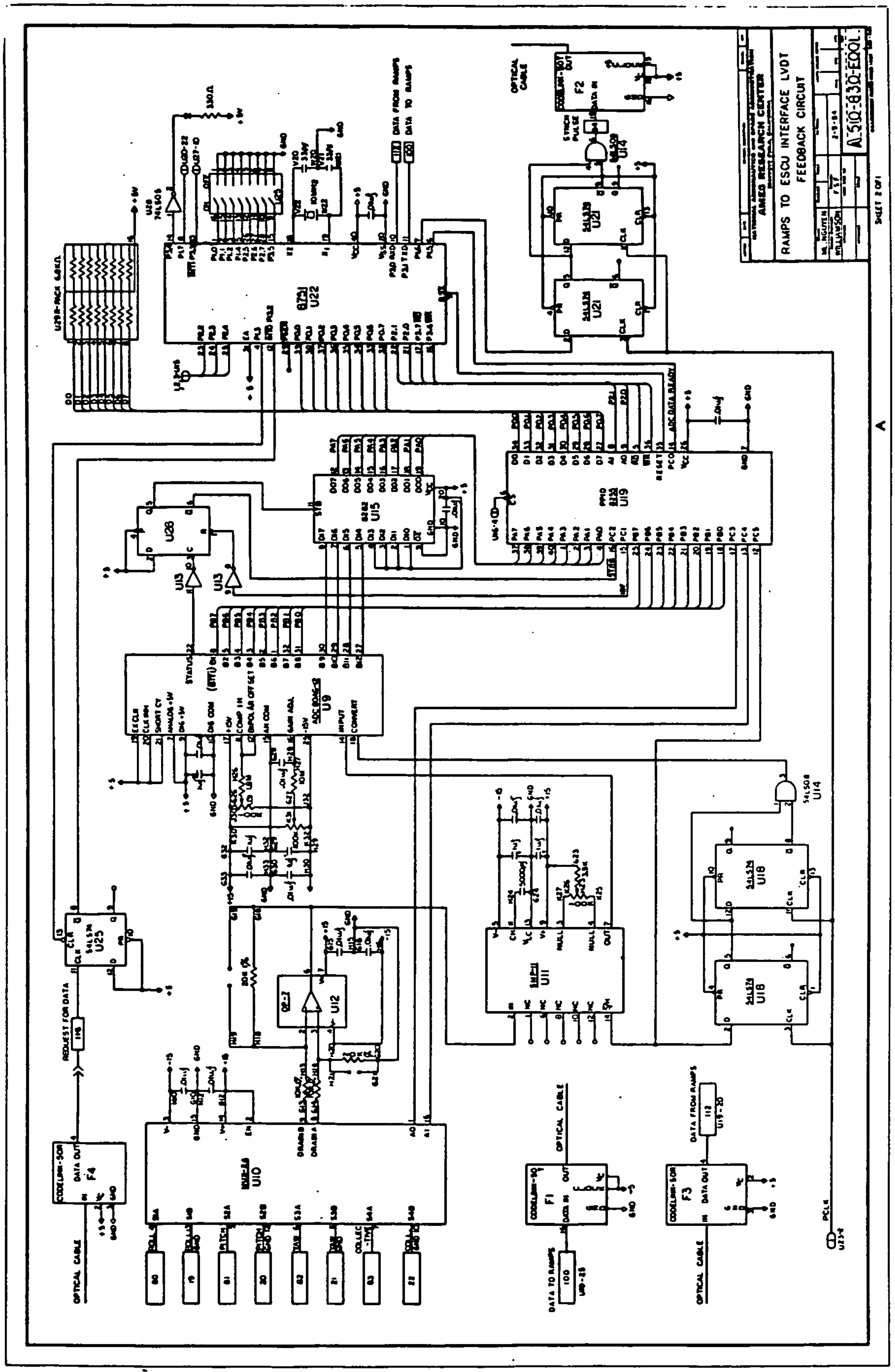
APPENDIX A

UFTCS/ESCU INTERFACE SCHEMATICS

PRECEDING PAGE BLANK NOT FILMED

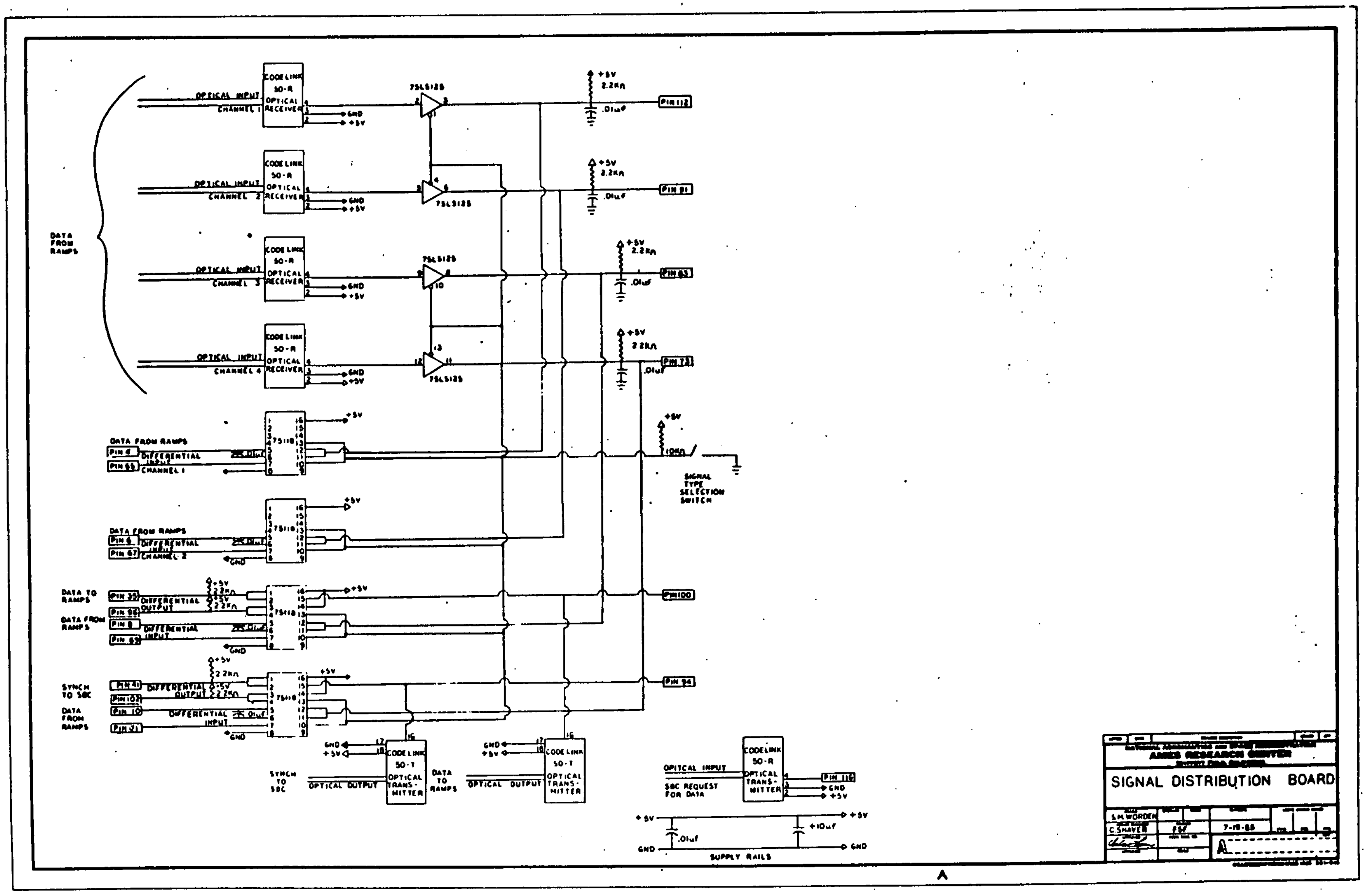


ORIGINAL PAGE IS
OF POOR QUALITY



25

ORIGINAL PAGE IS
OF POOR QUALITY



26

ORIGINAL PAGE IS
OF POOR QUALITY

APPENDIX B

UFTCS EMULATOR LISTING AND SCHEMATIC

SOFTWARE LISTING FOR THE [illegible] ENCODER

```
;***************************************************************************
;* This program will simulate the RAMPS flight control system.  The program *
;* will transmit the following message:                                     *
;*                                                                          *
;*   *************************************************************          *
;*   * Message * Subblock 1 * Roll * Roll * Pitch * Pitch * Yaw  * Yaw  *    *
;*   * Byte    * Count      * Low  * High * Low   * High  * Low  * High *    *
;*   * Count   *            * Byte * Byte * Byte  * Byte  * Byte * Byte *    *
;*   *************************************************************          *
;*                                                                          *
;*   ******************************************************                 *
;*   * Collective * Collective * Extra * Subblock 2 * Checksum *            *
;*   * Low        * High       * Data  * Count      *          *            *
;*   * Byte       * Byte       *       *            *          *            *
;*   ******************************************************                 *
;*                                                                          *
;*   The Message Byte Count is set to 15 and the Subblock 1 Count is 13.    *
;* The Roll, Pitch, Yaw, and Collective take up eight bytes of Subblock 1.  *
;* The Extra Data will use five bytes.                                      *
;*   Following the Extra Data is the Subblock 2 count.  This has been set to *
;* zero.  The Checksum has no use in the RAMPS system and has been set to   *
;* FFH.                                                                     *
;*   This program uses the serial port and timer 0 of the 8751 to transmit a *
;* message. The serial port is initialized to operate as a 9 bit UART with a *
;* baud rate of 156k.                                                       *
;*   A message will be transmitted by the serial port followed by about a   *
;* 1ms delay until the messasge is transmitted again.                       *
;*   The command axes will derive their inputs from a function generator    *
;* connected to the circuit.                                                *
;***************************************************************************
;
                                NAME RMPSIM
;
;
;                                 EQUATES
     PPIA        EQU        0000H          ;ADDRESS OF PPI PORT A
     PPIB        EQU        0100H          ;ADDRESS OF PPI PORT B
     CWR         EQU        0300H          ;ADDRESS OF PPI CONTROL WORD REGISTER
;
;                               DEFINE BITS
     TX_BUSY     BIT        3EH            ;TRANSMITTER AND TIMER 0 BUSY FLAG
;
                               ORG 0000H
     SJMP        START
;
                               ORG 000BH
;
;                   TIMER 0 INTERRUPT SERVICE ROUTINE
TZERO:           CLR        TCON.4         ;CLEAR BIT TO TURN OFF TIMER 0
                 CLR        TX_BUSY        ;CLEAR TIMER 0 BUSY FLAG
                 MOV        TH0,R2         ;RELOAD TIMER 0 WITH SELECTED
                 MOV        TL0,R1         ;UPDATE RATE
                 RETI
;
                               ORG 0023H
;
;                  SERIAL PORT INTERRUPT SERVICE ROUTINE
SRLPRT:          CLR        TI             ;CLEAR TRANSMIT INTERRUPT FALG
                 CLR        TX_BUSY        ;CLEAR TRANSMITER BUSY FLAG
                 RETI
;
```

```
                                ORG 0040H
START:          MOV        SP,#18H          ;INTIALIZE STACK POINTER, STARTS AT
                                            ;RO OF REGISTER BANK 3
                ACALL      INIT             ;GO TO INITIALIZATION ROUTINE
                MOV        C,P1.1           ;CHECK FOR ADC CALIBRATION
                JC         DOADC            ;IF SET GO TO CALIBRATE ADC
                MOV        C,P1.2           ;CHECK FOR SELECTED MESSAGE UPDATE RATE
                JC         _25MS            ;IF CHOSEN, USE 25MS UPDATE RATE
                MOV        C,P1.3           ;CHECK FOR SELECTED MESSAGE UPDATE RATE
                JC         _10MS            ;IF CHOSEN, USE 10MS UPDATE RATE
                MOV        C,P1.4           ;CHECK FOR SELECTED MESSAGE UPDATE RATE
                JC         FAST             ;IF CHOSEN, USE FASTEST UPDATE RATE
                MOV        R2,#5DH          ;IF NO SWITCHES SELECTED, USE 50MS
                MOV        R1,#3CH          ;UPDATE RATE
                JMP        GO
_25MS:          MOV        R2,#0AEH         ;USE R1 AND R2 TO HOLD COUNT FOR
                MOV        R1,#9DH          ;SELECTED UPDATE RATE (25MS)
                JMP        GO
_10MS:          MOV        R2,#0DFH         ;USE R1 AND R2 TO HOLD COUNT FOR
                MOV        R1,#73H          ;SELECTED UPDATE RATE (10MS)
                JMP        GO
FAST:           MOV        R2,#0E2H         ;USE R1 AND R2 TO HOLD COUNT FOR
                MOV        R1,#02AH         ;SELECTED UPDATE RATE (9.   MS)
GO:             MOV        TH0,R2           ;LOAD TIMER 0 WITH SELECTED
                MOV        TL0,R1           ;UPDATE RATE
                MOV        C,P2.3           ;CHECK FOR NUMBER OF EXTRA BYTES TO
                JC         _76BYTE          ;SEND, THIS DETERMINES MESSAGE LENGTH
                MOV        C,P2.4           ;CHECK FOR NUMBER OF EXTRA BYTES TO
                JC         _36BYTE          ;SEND, THIS DETERMINES MESSAGE LENGTH
                MOV        C,P2.5           ;CHECK FOR NUMBER OF EXTRA BYTES TO
                JC         _20BYTE          ;SEND, THIS DETERMINES MESSAGE LENGTH
                MOV        R3,#0AH          ;IF NO SWITCHES SET, MESSAGE COUNT = 10
                MOV        R4,#08H          ;SUBBLOCK 1 COUNT = 8
                MOV        R5,#00H          ;EXTRA BYTES = 0
                JMP        GOGO
_76BYTE:        MOV        R3,#4CH          ;IF SWITCH 1 SET, MESSAGE COUNT = 76
                MOV        R4,#4AH          ;SUBBLOCK 1 COUNT = 74
                MOV        R5,#42H          ;EXTRA BYTES = 66
                JMP        GOGO
_36BYTE:        MOV        R3,#24H          ;IF NO SWITCHES SET, MESSAGE COUNT = 36
                MOV        R4,#22H          ;SUBBLOCK 1 COUNT = 34
                MOV        R5,#1AH          ;EXTRA BYTES = 26
                JMP        GOGO
_20BYTE:        MOV        R3,#14H          ;IF NO SWITCHES SET, MESSAGE COUNT = 20
                MOV        R4,#12H          ;SUBBLOCK 1 COUNT = 18
                MOV        R5,#0AH          ;EXTRA BYTES = 10
GOGO:           SETB       IE.7             ;ENABLE INTERRUPTS
                JMP        XMIT_MSG         ;IF NOT SET START TRANSMITTING
                                            ;RAMPS MESSAGE
DOADC:          LJMP       ADCCAL           ;JUMP TO ADC CALIBRATION ROUTINE
```

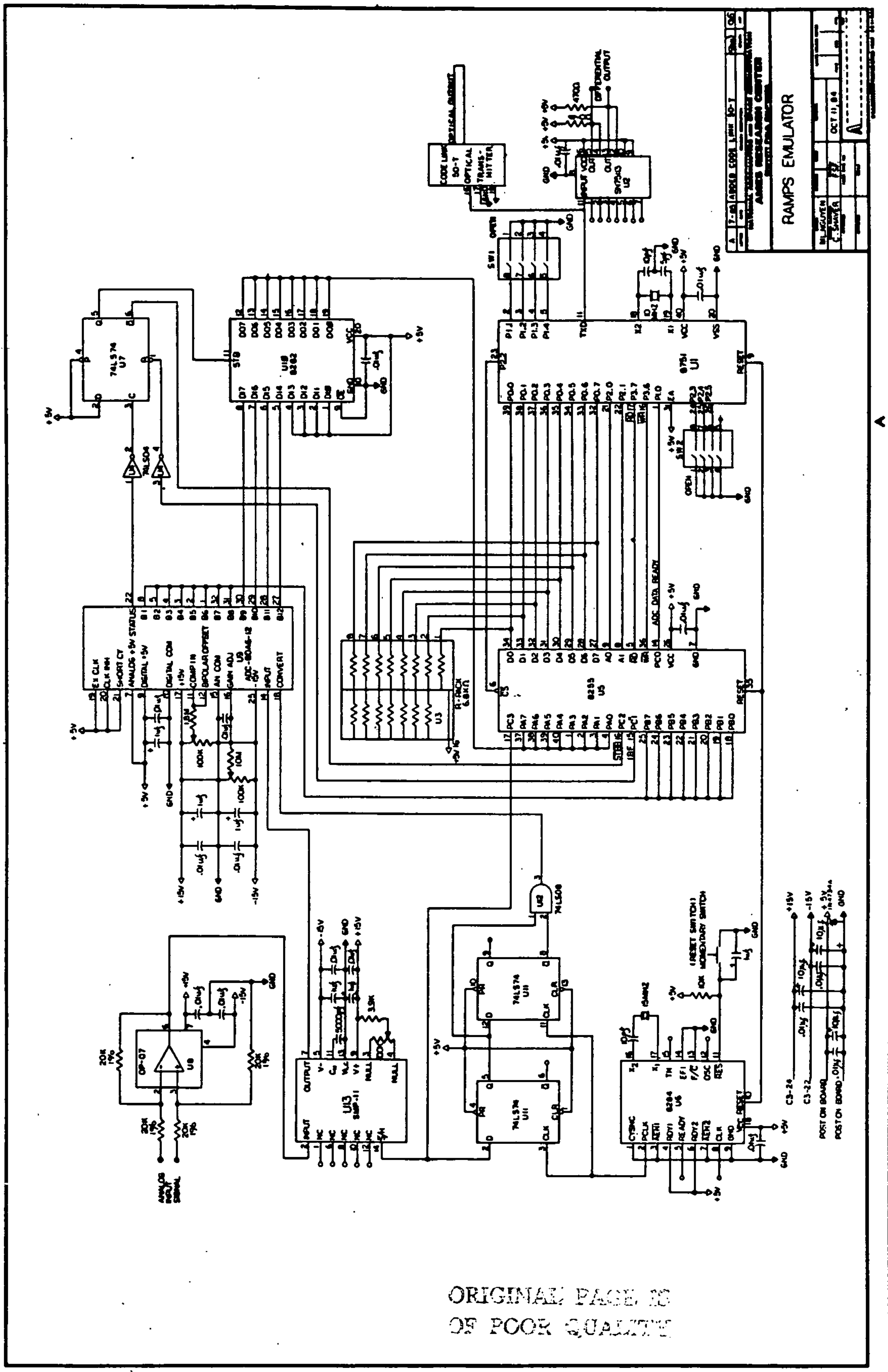
ORIGINAL PAGE IS
OF POOR QUALITY

```
;************************************************************************
;*   THIS IS THE INITIALIZATION ROUTINE FOR THE RAMPS SIMULATOR.  THIS  *
;* SECTION OF CODE WILL SET UP THE SERIAL PORT AND TIMER 0 ON THE 8751. THE *
;* ROUTINE ALSO INITIALIZES THE HARRIS PERIPHERAL INTERFACE (PPI).  THE 8751 *
;* AND THE PPI ARE INITIALIZED AS SHOWN BELOW:                          *
;*                                                                      *
;*                                   PPI                                *
;*                                                                      *
;*                          PORT A - BASIC INPUT                        *
;*                          PORT B - STROBED INPUT                      *
;*                          PORT C - HANDSHAKING AND CONTROL            *
;*                                                                      *
;*                                   8751                               *
;*                                                                      *
;*                SERIAL PORT - 156K BAUD, ODD PARITY (9 BIT UART)      *
;*                TIMER 0     - 13 BIT TIMER (MODE 0)                   *
;*                                                                      *
;*   THE INTERRUPTS FOR THE SERIAL PORT AND TIMER 0 HAVE BOTH BEEN SET TO *
;* HAVE HIGH PRIORITY.                                                  *
;************************************************************************
;
INIT:           MOV     DPTR,#CWR       ;MOVE ADDRESS OF PPI CONTROL REGISTER
                                        ;TO DATA POINTER
                MOV     A,#96H          ;PUT CONTROL WORD IN ACCUMULATOR
                MOVX    @DPTR,A         ;WRITE CONTROL WORD TO PPI
                MOV     A,#05H          ;SET PC2 IN PPI TO ENABLE INTE B
                MOVX    @DPTR,A         ;WRITE TO PPI, SETS PC2
                MOV     A,#06H          ;SET PC3 IN PPI FOR A/D CONVERT PULSE
                MOVX    @DPTR,A         ;ENABLE
                MOV     TMOD,#01H       ;PLACE TIMER 0 IN MODE 1 (16 BIT TIMER)
                MOV     SCON,#80H       ;PLACE SERIAL PORT IN 9 BIT UART MODE
                MOV     A,#00H          ;CLEAR ACCUMULATOR TO INITIALIZE R0
                MOV     R0,A            ;INITIALIZE R0 TO ZERO
                SETB    IE.4            ;ENABLE SERIAL PORT INTERRUPT
                SETB    IE.1            ;ENABLE TIMER 0 INTERRUPT
                SETB    IP.4            ;SET SERIAL PORT TO HIGHER PRIORITY
                SETB    IP.1            ;SET TIMER 0 TO HIGHER PRIORITY
                CLR     TX_BUSY         ;SET TRANSMITTER FLAG TO NOT BUSY
                CLR     PSW.4           ;CLEAR PSW.4 AND PSW.3 TO SELECT
                CLR     PSW.3           ;REGISTER BANK 0
                RET                     ;RETURN FROM SUBROUTINE
;
;************************************************************************
;*   THIS IS THE MAIN BLOCK OF CODE FOR THE RAMPS SIMULATOR.  THIS CODE  *
;* USES THE SERIAL PORT OF THE 8751 TO TRANSMIT A "RAMPS LIKE" MESSAGE.  *
;* THE FORMAT OF THE MESSASGE IS EXACTLY AS IT IS SPECIFIED IN THE BEGINNING *
;* OF THE PROGRAM.  IN SUMMARY, THE MESSAGE FORMAT IS:                  *
;*                                                                      *
;*   ***************************************************************    *
;*   * Message * Subblock 1 * Roll * Roll * Pitch * Pitch * Yaw  * Yaw  *  *
;*   * Byte    * Count      * Low  * High * Low   * High  * Low  * High *  *
;*   * Count   *            * Byte * Byte * Byte  * Byte  * Byte * Byte *  *
;*   ***************************************************************    *
;*                                                                      *
;*   ***********************************************************        *
;*   * Collective * Collective * Extra * Subblock 2 * Checksum *        *
;*   * Low        * High       * Data  * Count      *          *        *
;*   * Byte       * Byte       *       *            *          *        *
;*   ***********************************************************        *
;************************************************************************
;
```

```
XMIT_MSG:      MOV      A,R3             ;FORM MESSAGE BYTE COUNT IN ACCUMULATOR
               ACALL    PARITY           ;SET UP PARITY BIT - USE ODD PARITY
               SETB     TX_BUSY          ;SET TRANSMITTER BUSY FLAG TO BUSY
               MOV      SBUF,A           ;START TRANSMITTING MESSAGE BYTE COUNT
TXMT1:         JNB      TI,BUSY1         ;WAIT FOR TRANSMITTER TO FINISH
BUSY1:         JB       TX_BUSY,TXMT1    ;TRANSMITTING MESSAGE
               MOV      A,R4             ;FORM SUBBLOCK 1 COUNT IN ACCUMULATOR
               ACALL    PARITY           ;SET UP PARITY BIT - USE ODD PARITY
               SETB     TX_BUSY          ;SET TRANSMITTER BUSY FLAG TO BUSY
               MOV      SBUF,A           ;START TRANSMITTING SUBBLOCK 1 COUNT
TXMT2:         JNB      TI,BUSY2         ;WAIT FOR TRANSMITTER TO FINISH
BUSY2:         JB       TX_BUSY,TXMT2    ;TRANSMITTING MESSAGE
;
;*****************************************************************************
;*   THE A_TO_D ROUTINE USES A SAMPLE/HOLD WITH AN ANALOG TO DIGITAL         *
;* CONVERTER TO TRANSMIT THE 8 BYTES FOR NEEDED FOR THE FOUR COMMAND AXES.   *
;* EACH COMMAND AXIS IS MADE UP OF TWO BYTES.  THE LOW BYTE ALWAYS PRECEDES  *
;* THE HIGH BYTE.                                                            *
;*   THE ROUTINE BELOW IS LOOPED THROUGH EIGHT TIMES USING RO IN REGISTER    *
;* ZERO AS THE COUNTER FOR THE LOOP.                                         *
;*****************************************************************************
;
A_TO_D:        INC      RO               ;INCREMENT LOOP COUNTER
               MOV      DPTR,#CWR        ;LOAD ADDRESS OF CONTROL WORD REGISTER
                                         ;INTO THE DATA POINTER
               MOV      A,#07H           ;SET PC3 OF PPI TO START A/D
               MOVX     @DPTR,A          ;CONVERSION
               MOV      A,#06H           ;RESET PC3 OF PPI TO REMOVE CONVERSION
               MOVX     @DPTR,A          ;COMMAND
               MOV      DPTR,#PPIA       ;LOAD ADDRESS OF PPI PORT A INTO THE
                                         ;DATA POINTER
               JNB      P1.0,$           ;WAIT FOR A/D CONVERSION TO FINISH
               MOVX     A,@DPTR          ;READ LOW BYTE OF COMMAND FROM PPI
               ACALL    PARITY           ;SET UP PARITY BIT - USE ODD PARITY
               SETB     TX_BUSY          ;SET TRANSMITTER BUSY FLAG TO BUSY
               MOV      SBUF,A           ;START TRANSMITTING LOW BYTE OF COMMAND
TXMT3:         JNB      TI,BUSY3         ;WAIT FOR TRANSMITTER TO FINISH
BUSY3:         JB       TX_BUSY,TXMT3    ;TRANSMITTING MESSAGE
               MOV      DPTR,#PPIB       ;LOAD ADDRESS OF PPI PORT B INTO THE
                                         ;DATA POINTER
               MOVX     A,@DPTR          ;READ HIGH BYTE OF COMMAND FROM PPI
               ACALL    PARITY           ;SET UP PARITY BIT - USE ODD PARITY
               SETB     TX_BUSY          ;SET TRANSMITTER BUSY FLAG TO BUSY
               MOV      SBUF,A           ;TRANSMIT HIGH BYTE OF COMMAND
TXMT4:         JNB      TI,BUSY4         ;WAIT FOR TRANSMITTER TO FINISH
BUSY4:         JB       TX_BUSY,TXMT4    ;TRANSMITTING MESSAGE
               CJNE     RO,#04H,A_TO_D   ;IF RO IS NOT 8, THEN ALL THE COMMAND
                                         ;BYTES HAVE NOT BEEN TRANSMITTED
                                         ;CONTINUE UNTIL DONE
               MOV      A,#00H           ;CLEAR ACCUMULATOR TO RESET RO
               MOV      RO,A             ;RESET RO FOR NEXT LOOP
```

```
;
;*************************************************************************
;*   THIS ROUITNE WILL TRANSMIT THE EXTRA DATA USED IN THE RAMPS SIMULATED    *
;* MESSAGE.  RO OF REGISTER BANK O IS USED AS A COUNTER IN THIS LOOP.  FIVE   *
;* BYTES OF DATA ARE TRANSMITTED AS EXTRA DATA.  THE DATA WILL BE THE CURRENT *
;* VALUE OF THE LOOP COUNTER (RO).                                            *
;*************************************************************************
;
                MOV     A,R5            ;SEE IF THE EXTRA BYTE COUNT IS
                JZ      NOBYTES         ;ZERO AND JUMP AROUND IF IT IS
                MOV     A,R5            ;USE R6 AS THE LOOP COUNTER
                MOV     R6,A
EXDATA:         INC     RO              ;RO ACTS AS EXTRA DATA BYTE
                MOV     A,RO            ;PREPARE TO TRANSMIT EXTRA DATA
                ACALL   PARITY          ;SET UP PARITY BIT - USE ODD PARITY
                SETB    TX_BUSY         ;SET TRANSMITTER BUSY FLAG TO BUSY
                MOV     SBUF,A          ;START TRANSMITTING EXTRA DATA
TXMT5:          JNB     TI,BUSY5        ;WAIT FOR TRANSMITTER TO FINISH
BUSY5:          JB      TX_BUSY,TXMT5   ;TRANSMITTING MESSAGE
                DEC     R6              ;DECREMENT THE LOOP COUNTER
                MOV     A,R6            ;IF ACCUMULATOR IS NOT ZERO THEN ALL
                                        ;EXTRA DATA HAS NOT BEEN TRANSMITTED
                JNZ     EXDATA          ;CONTINUE UNTIL DONE
NOBYTES:        MOV     A,#OOH          ;CLEAR ACCUMULATOR TO RESET RO
                MOV     RO,A            ;RESET RO AND TRANSMIT AS
                                        ;SUBBLOCK 2 COUNT
                ACALL   PARITY          ;SET UP PARITY BIT - USE ODD PARITY
                SETB    TX_BUSY         ;SET TRANSMITTER BUSY FLAG TO BUSY
                MOV     SBUF,A          ;START TRANSMITTING SUBBLOCK 2 COUNT
TXMT6:          JNB     TI,BUSY6        ;WAIT FOR TRANSMITTER TO FINISH
BUSY6:          JB      TX_BUSY,TXMT6   ;TRANSMITTING MESSAGE
                MOV     A,#OFFH         ;LOAD ACCUMULATOR WITH DUMMY CHECKSUM
                ACALL   PARITY          ;SET UP PARITY BIT - USE ODD PARITY
                SETB    TX_BUSY         ;SET TRANSMITTER BUSY FLAG TO BUSY
                MOV     SBUF,A          ;START TRANSMITTING DUMMY CHECKSUM
TXMT7:          JNB     TI,BUSY7        ;WAIT FOR TRANSMITTER TO FINISH
BUSY7:          JB      TX_BUSY,TXMT7   ;TRANSMITTING MESSAGE
;
;*************************************************************************
;*   THE FOLOWING CODE LOADS TIMER O WITH A VALUE THAT WILL MAKE IT TIME OUT  *
;* IN A LITTLE OVER 1MS.  AFTER THE TIMER TIMES OUT, THE NEXT MESSAGE IS      *
;* IS TRANSMITTED.                                                            *
;*************************************************************************
;
                SETB    TX_BUSY         ;SET TIMER BUSY FLAG TO BUSY
                SETB    TCON.4          ;START TIMER O
TMR1:           JNB     TCON.5,TBUSY1   ;WAIT FOR TIMER TO TIME OUT BEFORE
TBUSY1:         JB      TX_BUSY,TMR1    ;TRANSMITTING THE NEXT MESSAGE
                LJMP    XMIT_MSG        ;JUMP TO START THE NEXT MESSAGE
;
;*************************************************************************
;*   THE FOLLOWING SHORT ROUTINE ADDS THE PARITY BIT TO THE MESSAGE.  THE     *
;* PARITY BIT IS THE 9TH BIT TRANSMITTED AND IS PLACED IN THE SERIAL PORT     *
;* CONTROL REGISTER - BIT 3 (SCON.3).  ODD PARITY IS USED.                    *
;*************************************************************************
;
PARITY:         MOV     C,P             ;MOVE PARITY BIT TO CARRY FLAG
                CPL     C               ;COMPLEMENT CARRY FOR ODD PARITY
                MOV     SCON.3,C        ;PLACE IN SCON.3 AS 9TH BIT
                RET                     ;RETURN FROM SUBROUTINE
```

```
;
;*************************************************************************
;*          ANALOG TO DIGITAL CONVERTER / SAMPLE AND HOLD CALIBRATION   *
;*                                                                       *
;*    THIS ROUTINE REQUIRES A PRECISION VOLTAGE SUPPLY BE CONNECTED TO THE *
;* OP-AMP INPUT OF THE RAMPS SIMULATOR.                                  *
;*    THE PROCEDURE IS AS FOLLOWS:                                       *
;*                                                                       *
;*    A) OFFSET ADJUSTMENT                                               *
;*       1. ASSERT SWITCH 1 OF DIPSWITCHES, RESET SIMULATOR AND ALLOW TO WARM *
;*          UP 10 MINUTES BEFORE ATTEMPTING A CALIBRATION.               *
;*       2. CONNECT THE PRECISION VOLTAGE SOURCE TO THE OPAMP INPUT OF THE *
;*          SIMULATOR                                                    *
;*       3. SET THE PRECISION VOLTAGE SOURCE TO -9.999V                  *
;*       4. ADJUST THE SAMPLE/HOLD POTENTIOMETER UNTIL THE SAMPLE/HOLD OUTPUT *
;*          OUTPUT READS -9.999V                                         *
;*       5. ADJUST THE ADC OFFSET POTENTIOMETER UNTIL THE ADC OUTPUT READS *
;*          FFEH                                                         *
;*                                                                       *
;*    B) GAIN ADJUSTMENT                                                 *
;*       1. SWITCH THE POLARITY OF THE PRECISION VOLTAGE SOURCE SO THAT THE *
;*          OUTPUT OF THE SOURCE IS NOW +9.999V                          *
;*       2. ADJUST THE ADC GAIN POTENTIOMETER UNTIL THE ADC OUTPUT READS  *
;*          000H                                                         *
;*************************************************************************
;
ADCCAL:         CLR     IE.7            ;DISABLE INTERRUPTS; THEY AREN'T NEEDED
                MOV     DPTR,#CWR       ;LOAD ADDRESS OF PPI CONTROL WORD
                                        ;REGISTER INTO THE DATA POINTER
                MOV     A,#07H          ;MOVE WORD TO SET PC3 INTO ACCUMULATOR
                MOVX    @DPTR,A         ;WRITE TO PPI, SENDS CONVERT PULSE
                MOV     A,#06H          ;PUT 6 IN ACCUMULATOR TO TURN OFF
                MOVX    @DPTR,A         ;CONVERT PULSE (RESET PC3)
                JNB     P1.0,$          ;WAIT FOR CONVERSION TO GET DONE
                MOV     DPTR,#PPIB      ;LOAD ADDRESS OF PPI PORT B INTO THE
                                        ;DATA POINTER
                MOVX    A,@DPTR         ;PERFORM DUMMY READ TO CLEAR IBF
                                        ;PIN ON THE PPI
                SJMP    ADCCAL          ;CONTINUE SAMPLING AND HOLDING UNTIL
                                        ;SYSTEM IS RESET
END
```



ORIGINAL PAGE IS
OF POOR QUALITY

1. Report No. NASA TM 88236	2. Government Accession No.	3. Recipient's Catalog No.

4. Title and Subtitle DEVELOPMENT OF AN INTERFACE FOR AN ULTRARELIABLE FAULT-TOLERANT CONTROL SYSTEM AND AN ELECTRONIC SERVO-CONTROL UNIT	5. Report Date September 1986
	6. Performing Organization Code
7. Author(s) Charles Shaver and Michael Williamson	8. Performing Organization Report No. A-86196
	10. Work Unit No.
9. Performing Organization Name and Address Ames Research Center Moffett Field, CA 94035	11. Contract or Grant No.
	13. Type of Report and Period Covered Technical Memorandum
12. Sponsoring Agency Name and Address National Aeronautics and Space Administration Washington, DC 20546	14. Sponsoring Agency Code 505-34-01

15. Supplementary Notes

Point of contact: Charles Shaver, Ames Research Center, MS 210-5, Moffett Field, CA 94035. (415)694-5941 or FTS 464-5941

16. Abstract

The NASA Ames Research Center sponsors a research program for the investigation of Intelligent Flight Control Actuation Systems. The use of artificial intelligence techniques in conjunction with algorithmic techniques for autonomous, decentralized fault management of flight-control actuation systems will be explored under this program. This paper documents the design, development, and operation of the interface and emulator equipment for laboratory investigations of this research program. The interface, architecturally based on the Intel 8751 microcontroller, is an interrupt-driven system designed to receive a digital message from an ultrareliable fault-tolerant control system (UFTCS). The interface links the UFTCS to an electronic servo-control unit, which controls a set of hydraulic actuators. It was necessary to build a UFTCS emulator (also based on the Intel 8751) to provide signal sources for testing the equipment.

This paper discusses the conversion of the 8-byte message (characteristic command for the four control axes of a helicopter) to the appropriate byte length.

17. Key Words (Suggested by Author(s)) Interface; Fault-tolerant control system; Actuators, Microcontroller; Redundancy	18. Distribution Statement Unlimited Subject category - 09

19. Security Classif. (of this report) Unclassified	20. Security Classif. (of this page) Unclassified	21. No. of Pages 37	22. Price* A03

*For sale by the National Technical Information Service, Springfield, Virginia 22161